Encoding

Suresh Podeti

3 min read · May 21, 2023

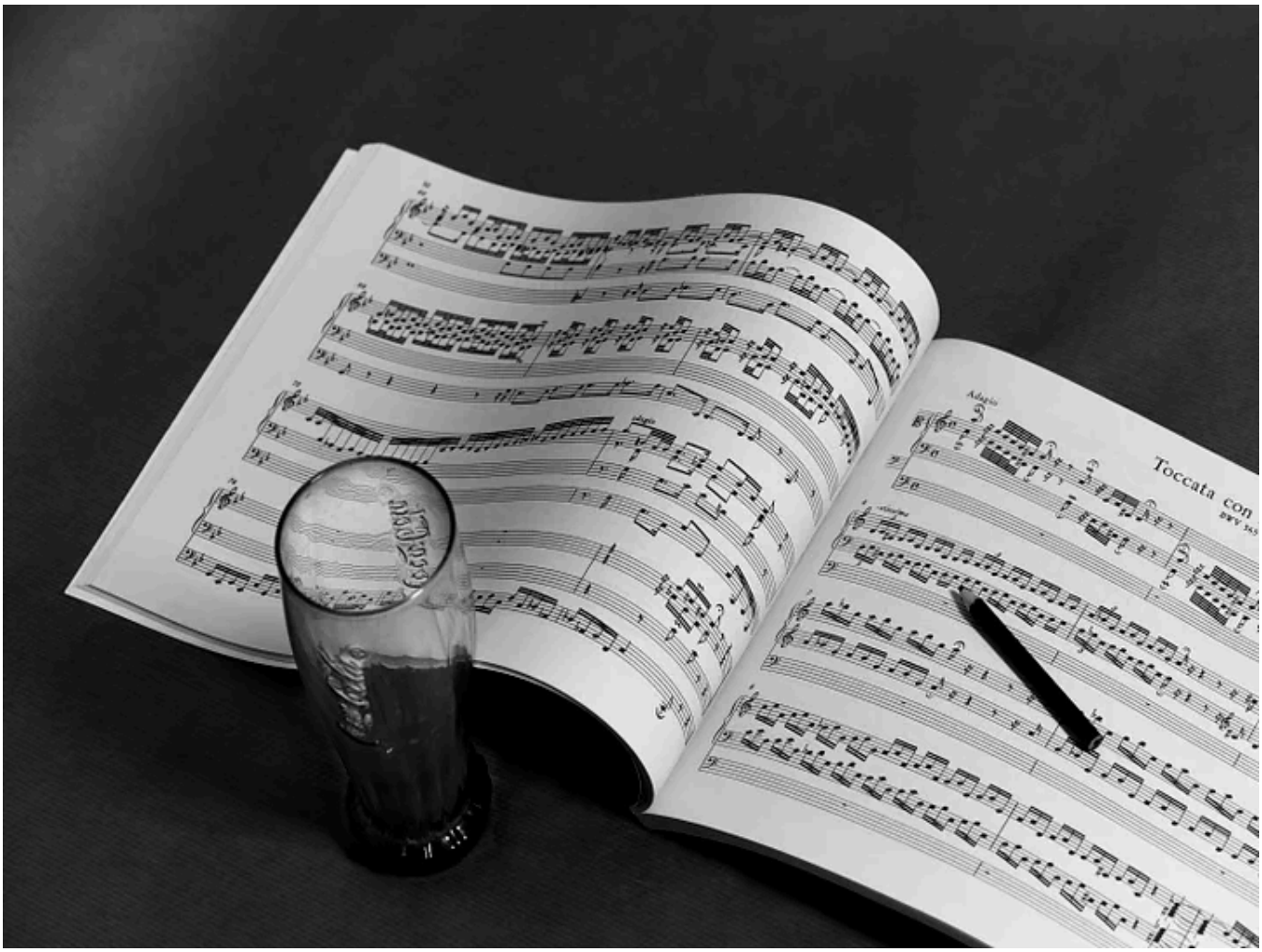

Photo by Xavier von Erlach on Unsplash

Server-side application loads **database query result into an in-memory data structure** like objects, but needs to be converted to a different format to sent over the network. Usually, data sent over network is self-contained **sequence of bytes** like JSON document, and are different from the data structures that are normally used in memory.

We need some kind of **translation** between the two representations, and translation from the **in-memory representation** to a **byte sequence** is called **encoding** (also known as serialisation or marshalling), and the reverse is called **decoding** (parsing, deserialisation, un-marshalling). Byte sequence are optimised for transport, read, and storage.

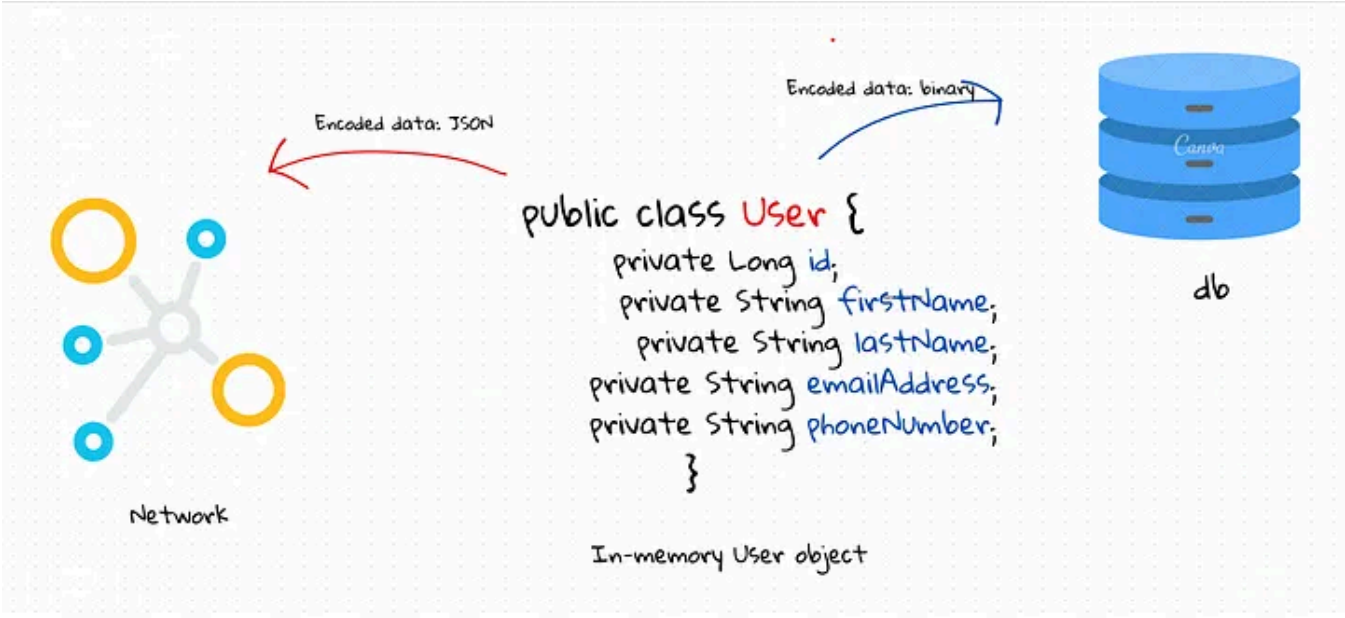

Fig 1.0: In-memory object is converted and sent through network, and stored in db

Many programming languages come with built-in support for encoding in-memory objects into byte sequences. For example, **Java has java.io.Serializable, Python has pickle** etc.. They have problems like: Encoding is often tied to a particular programming language, and reading the data in another language is very difficult.

```
import pickle

user = {
    "id": 1,
    "first_name": "Suresh",
    "last_name": "Podeti",
    "email": "suri.podeti7@gmail.com",
    "phone": "9xxxxxxxx2"
}
#
result = pickle.dumps(user)
print(result)
# Outputs: b'\x80\x04\x95m\x00\x00\x00\x00\x00\x00\x00}\x94(\x8c\x02id\x94K\x01
```

Pickle encoded data is only read by Python's pickle library, so this kind of encoding is tightly coupled with programming language.

Types of Encoding Formats

There are two types of encoding formats:

1. Textual formats
2. Binary formats

Textual Formats

Fig 2.0: Somewhat human-readable textual encoding formats

Textual formats are **somewhat human-readable.** Some of the common formats are **JSON, XML and CSV.**

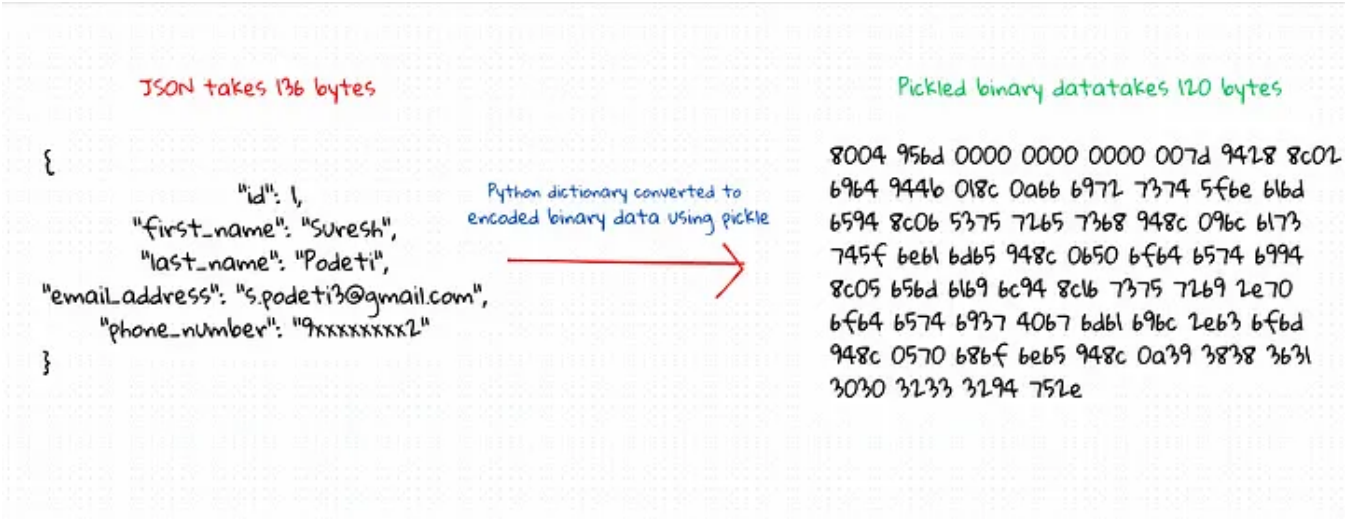

Fig 3.0: Python dictionary data converted to binary using pickle

They are **somewhat human-readable**, however, they take a **lot of space** compared to binary formats, and they can result into different problems mentioned below:

1. **Ambiguity around the encoding of numbers** — XML and CSV formats, we cannot distinguish between strings and numbers. Eg. we can not tell if 30 is string or number. JSON can distinguish between string and numbers, but **cannot distinguish between integers and floating numbers** and doesn't specify a **precision** when **dealing with large numbers.**

```
let a = '{"tokenId": 99999999999999999999999999}';
let b = '{"tokenId": 99999999999999999999999999.7}';

let parsed_a = JSON.parse(a)
let parsed_b = JSON.parse(b)

console.log(parsed_a.tokenId == parsed_b.tokenId)
// Outputs: true; not distinguishing b/w integer and floating number
```

```
let c = '{"tokenId": 99999999999999999999999999.7}';

let parsed_c = JSON.parse(c)


console.log(parsed_c.tokenId)

//Outputs: 10^28; precision lost.
```

2. **Schema** — There is optional schema support for both XML, and JSON. JSON based tools don't bother using schemas, and CSV doesn't contain any schema, leaving it to the application to define the meaning of each row and column.

Binary Encoding

For data that is used only **internally within your organization**, you could choose a format that is **more compact or faster to parse, because** binary encoding are lighter solutions but lose the ability to be read by humans. Binary encoding varies in parsing strategies and size performance. Examples are **Protocol Buffer, Thrift, and Avro.**

All three provide efficient, **cross-language serialization** of data using a schema and **have code generation tools.** All three support schema evolution by ensuring both backward and forward compatibility.

Read more about backward and forward compatibility on my previous blog:

We will explore binary encoding and how it supports schema evolution in my next article.

References:

1. O'Reilly designing data-intensive applications by Martin Kleppmann, Chapter 4: Encoding and Evolution

Encoding Serialisation Database Software Development Networking

Edit profile

Written by Suresh Podeti

1.2K Followers

Aspiring Software Architect | SDE III (Golang) @JungleeGames| Ex-Byju's | Co-founder @Rupant Tech. | Ex-Synopsys | I.I.T Kharagpur | 7+ Years of Experience